Introduction

Useful information

- My name: Giorgio Metta
- My email: pasa@liralab.it
- Office/Lab: 010 71781 411
- Where: LIRA-Lab, Villa Bonino, Ground Floor (su appuntamento)
- Web site: http://www.liralab.it/os
- Mailing list: os@liralab.it

Outline of the course

- Processes, threads, scheduling
- IPC
- Memory management
- I/O
- Filesystem
- Embedded systems

- The exam consists of:
  - 1 problem set
  - C++ programming 1/3
  - 1 oral exam:
    - Theory and short exercises 2/3

Background

- Required
  - Programming C/C++

- Helpful
  - Linux/Unix, Windows

- Main idea is to learn, so, don't freak out even if it might seem hard!

References


This slide is intentionally left blank
Operating system

- What’s inside the computer?
  - Layers:

<table>
<thead>
<tr>
<th>Web browser</th>
<th>Banking system</th>
<th>Airline reservation</th>
<th>application programs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Compilers</td>
<td>Editors</td>
<td>Command interpreter (shell)</td>
<td>system programs</td>
</tr>
<tr>
<td>Operating system</td>
<td>Machine language</td>
<td>Microarchitecture</td>
<td>hardware</td>
</tr>
<tr>
<td>Physical device</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Meaning of the layers

- Physical devices: self explaining
- Microarchitecture: define data path within the microprocessor (using registers) sometimes using a microprogram
- Machine language/Assembly language: instruction set (e.g. 50-300 instructions)

Where does the OS start?

- Kernel mode
  - Supervisor mode
  - Hardware protection (on modern microprocessors)
  - All instructions allowed
  - Timer interrupt handler
- User mode
  - Compiler, editor, web browser
  - Certain instructions not allowed

Example: microkernel OS

Microkernel RTOS. In QNX Neutrino, only the most fundamental OS primitives (e.g. signals, timers, scheduling) are handled by the kernel itself. All other components – drivers, file systems, protocol stacks, user applications – run outside the kernel as separate, memory-protected processes.

Operating system’s job

- Manage the hardware (all the devices)
- Provide user programs with simpler interface to the hardware (extended machine)

Example: floppy drive

- Specific chip (NEC PD765)
- 16 different commands
- Load between 1 and 9 bytes into a device register
- Read/Write require 13 parameters packed into 9 bytes
- Reply from the device consists of 7 bytes (23 parameters)
- Control of the motor (on/off)
Abstraction

• Better to think in terms of files with names rather than specific floppy drive commands
• Other unpleasant hardware:
  – Interrupts
  – Timers
  – Memory management
  – …
• Extended or virtual machine

OS as resource manager

• Allocation of resources:
  – Processors, memory, I/O devices among a set of programs competing for them
• Example: allocating the printer
  – Buffering output rather than just print at random
• Multiple users: sharing of resources and avoid conflicts (share vs. security)

Sharing

• Time and space multiplexing
• Multiplexing in time: e.g. printer, processor
  – Print one job at a time
• Multiplexing in space: e.g. memory, disks
  – Divide memory among many processes

Computer hardware

• Processors
• Memory
• I/O devices
• Buses

Processors

<table>
<thead>
<tr>
<th>Registers</th>
</tr>
</thead>
<tbody>
<tr>
<td>Program counter (PC): next instruction</td>
</tr>
<tr>
<td>Stack pointer (SP): stack in memory</td>
</tr>
<tr>
<td>Program Status Word (PSW): condition bits (e.g. kernel vs. user mode)</td>
</tr>
<tr>
<td>Base register: relocation of executables</td>
</tr>
</tbody>
</table>

System call

| SW interrupt |
| From User to Kernel mode |

Complexity of the CPU HW

| Pipeline architecture |
| Superscalar |

Memory

• Ideally...
  – Extremely fast (faster than the CPU in executing an instruction)
  – Abundantly large
  – Dirt cheap
Memory (for real)

Typical access time | Size
--- | ---
1 nsec | < 1M
2 nsec | 1-4G
10 nsec | 5-100G
100 nsec | > 20G

OS 2007-08

Memory cntd.

- Registers: typical 32 in a 32 bit CPU
- Cache: divided into cache lines (64 bytes each)
  - Cache hit – no main memory access, no bus involvement
  - Cache miss – costly
- Main memory
- Disk (multiple plates, heads, arms)
  - Logical structure: sectors, tracks, cylinders
- Magnetic tape: backup, cheap, removable

OS 2007-08

Multiple programs in memory

- Base and Limit register
- Hardware support for relocation and multiple programs in memory

```
Fetch: Instruction
if (PC < Limit) Fetch(PC + Base) else Troubleshoot(SigFault)

if (Addr < Limit) Fetch(Addr + Base) else Troubleshoot(SigFault)
```

OS 2007-08

DLL’s (in principle)

- Requires an MMU with multiple Base/Limit register pairs

OS 2007-08

Memory Management Unit

- Managing the MMU is one of the OS tasks:
  - Balancing context switches since they impact on performances: e.g. MMU registers have to be saved, cache emptied, etc.

OS 2007-08

I/O devices

- Usually a controller + the actual device
  - For example: a disk controller may hide the details of driving the arm and heads to the appropriate location to read a certain piece of data
  - Sometimes the controller is a small embedded microprocessor in itself
- The interface to the OS is somewhat standardized:
  - IDE disk drives conform to a standard
- Device driver: a piece of the OS. Device drivers run in kernel mode since they have to access I/O instructions and device registers

OS 2007-08
**Device drivers**

1. Unix. Compiled and linked with the kernel (although Linux supports dynamic loading of DD)
2. Windows. An entry into an OS table. Loaded at boot
3. Dynamic. USB, IEEE1394 (firewire). At boot time the OS detects the hardware, finds the DD, and loads them

**I/O registers**

- E.g. small number of registers used to communicate
- Memory mapped: the registers appear at particular locations within the OS address space
- I/O instructions: some CPUs have special privileged (kernel mode) I/O instructions (IN/OUT). Registers are mapped to special locations in I/O space

**Ways of doing I/O**

1. Polling
2. Interrupt based
3. DMA

**Polling**

- User makes a system call
- OS calls DD
- DD talks to device, prepares I/O, starts I/O and sits waiting (busy waiting) for I/O completion

- *Busy waiting* means that the CPU is busy polling a flag

**Interrupt**

- A piece of hardware called “interrupt controller”

1. CPU issues the I/O request via the device driver
2. On termination the device signals the CPU’s interrupt controller (if the interrupt controller is not busy servicing another higher priority interrupt)
3. If the interrupt can be handled then the controller asserts a pin on the CPU
4. The interrupt controller puts the address of the device into the bus

**Interrupt (cntd.)**

- When the CPU decides to take the interrupt:
  - Stores registers (push them into the stack)
  - Switches into kernel mode
  - Uses the device’s address to index a table (interrupt vector)
  - Calls the handler contained at the location located in the interrupt vector
  - Once the handler is executed it returns from the handler by popping the registers from the stack
Direct Memory Access DMA

- Yet another piece of hardware: DMA controller
  - Communication between memory and device can be carried out by the DMA controller with little CPU intervention
  - When the DMA is completed the controller asserts an interrupt as before

Buses

- Multiple buses (cache, local, memory, PCI, USB, IDE...)
- OS must be aware of all of them to manage things appropriately
- Plug&Play – dynamic allocation of I/O and memory addresses (BIOS code)

Concepts

- Processes
- Deadlocks
- Memory management
- I/O
- Files
- Security
- ...

The Shell

- Unix command interpreter (or similarly the "command" in windows)
- Clearly, it's not part of the OS

Processes

- Associated with each process:
  - Address space (program + data + stack)
  - Entry into the process table (a list of processes)
  - Set of registers (e.g. PC, PSW, etc.)
  - MMU status, registers
- Processes can be created, terminated, signaled (SW interrupt)
- They form a tree (a hierarchy) on some systems
- Process cooperation is obtained by means of IPC (inter-process communication) mechanisms
- Processes start with the privileges of the user who starts them
**ps (process status) command**

- Process ID
- Parent ID
- Owner UID
- Starting time
- Name

**Deadlocks**

- Two or more processes mutually requesting the same set of resources
- Example: two processes trying to use simultaneously a tape and CD burner in reverse order

**Memory management**

- Virtual memory
  - Allowing processes requesting more memory than the computer main memory to run
  - Swap space/swapping. Storing some of the process’ memory in the disk

**Files**

- Concept of directory (group files together)
- A tree-like structure similar to the process hierarchy
- A file is specified by its *path* name
  - E.g., `/usr/bin/ps`
- In UNIX there’s a *root* directory (`/`)
  - Windows has a root for each drive: A:, B:, C:, etc.
- Working directory (a process property)
  - Where path not beginning with slash are looked for
- Interface between OS and program code is through a small integer called *file descriptor*

**Special file**

- A device driver gets a special entry into the file system (usually under `/dev`)
- Block special files
  - Randomly addressable blocks: a disk
- Character special files
  - A stream of character data: modem, printer
Special file (ctnd.)

Security

Pipe
- It's a sort of pseudofile
- Allows connecting two processes as they were issuing read/write system calls to a regular file

Pipe example

System calls

This slide is intentionally left empty
System calls

count = read(fd, buffer, nbytes);

1. Push nbytes into the stack
2. Push buffer into the stack
3. Push fd into the stack
4. Library calls read
5. Put sys call code into register
6. Trap to kernel
7. Examines the call code, query table
8. Call handler, execute read code
9. Return to caller (maybe)
10. Pop stack (i.e. increment SP)
11. Continue execution
**lseek**

\[ \text{position} = \text{lseek}(\text{fd}, \text{offset}, \text{whence}) \]

- Random access to a file
- Imagine the file as accessed through a pointer
- \text{lseek} moves the pointer

**Directory (in UNIX)**

- Each file is identified by an \textit{i-number}
- The \textit{i-number} is an index into a table of \textit{i-nodes}
- A directory is a file containing a list of \textit{i-number} – ASCII name

**Link**

- Called a shortcut in some versions of Windows

<table>
<thead>
<tr>
<th>Function</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>call</td>
<td>name</td>
</tr>
<tr>
<td>open</td>
<td>note</td>
</tr>
</tbody>
</table>

```bash
link("/usr/jim/note", "usr/ast/note")
```

**Win32 API**

- Different philosophy
- Many calls (API – Application Program Interface), not all of them are actually system calls
- GUI included into the API (in comparison X-Windows is all user level code)

**Example of Win32**

<table>
<thead>
<tr>
<th>Function</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>Get the current time</td>
<td>GetLocalTime</td>
</tr>
<tr>
<td>Kill</td>
<td>Nonekill</td>
</tr>
<tr>
<td>chmod</td>
<td>Nonechmod</td>
</tr>
<tr>
<td>Change the current working directory</td>
<td>SetCurrentDirectory</td>
</tr>
<tr>
<td>chdir</td>
<td>Nonechdir</td>
</tr>
<tr>
<td>umount</td>
<td>Nonemount</td>
</tr>
<tr>
<td>Destroy an existing file</td>
<td>DeleteFile</td>
</tr>
<tr>
<td>unlink</td>
<td>Nonelink</td>
</tr>
<tr>
<td>Remove an empty directory</td>
<td>RemoveDirectory</td>
</tr>
<tr>
<td>Create a new directory</td>
<td>CreateDirectory</td>
</tr>
<tr>
<td>Get various file attributes</td>
<td>GetFileAttributeEx</td>
</tr>
<tr>
<td>Move the file pointer</td>
<td>SetFilePointer</td>
</tr>
<tr>
<td>Write data to a file</td>
<td>WriteFile</td>
</tr>
<tr>
<td>Read data from a file</td>
<td>ReadFile</td>
</tr>
<tr>
<td>Close a file</td>
<td>CloseHandle</td>
</tr>
<tr>
<td>Create a file or open an existing file</td>
<td>CreateFile</td>
</tr>
<tr>
<td>Terminate execution</td>
<td>ExitProcess</td>
</tr>
<tr>
<td>CreateProcess does the job</td>
<td>Noneexecve</td>
</tr>
<tr>
<td>Can wait for a process to exit</td>
<td>WaitForSingleObject</td>
</tr>
<tr>
<td>Create a new process</td>
<td>CreateProcess</td>
</tr>
</tbody>
</table>

**Operating system structure**

- Monolithic systems
- Layered systems
- Virtual machines
- Exokernels
- Client-Server model
Monolithic systems

• The “big mess”
• No organized structure
• A bit of structure anyway:
  – System calls require parameters in a well defined place (e.g. the stack)
  – Three layers:
    • Application program
    • Service procedures
    • Helper procedures

Layered systems

• Each layer relies only on services provided by lower level layers

<table>
<thead>
<tr>
<th>Layer</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>User/operator</td>
</tr>
<tr>
<td>2</td>
<td>User programs</td>
</tr>
<tr>
<td>3</td>
<td>I/O management</td>
</tr>
<tr>
<td>4</td>
<td>Operator process communication</td>
</tr>
<tr>
<td>5</td>
<td>Memory and disk management</td>
</tr>
<tr>
<td>6</td>
<td>Processor allocation and multiprogramming</td>
</tr>
</tbody>
</table>

Virtual machines

• Timesharing provides:
  – Multiprogramming
  – Extended machine
• Decouple the two functions:
  – Virtual machine monitor (a SW layer)
  – It does the multiprogramming providing a “simulation” of the bare HW
• On top of the monitor any compatible OS could be run
• Also the Pentium (8086 mode, running DOS applications) and Java VM provide a similar mechanism (slightly different though)

Virtual machines

<table>
<thead>
<tr>
<th>syscall</th>
<th>Applications</th>
<th>I/O instr</th>
</tr>
</thead>
<tbody>
<tr>
<td>VM/70</td>
<td>CMS</td>
<td>CMS</td>
</tr>
<tr>
<td>370 Bare Hardware</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Exokernel

• Each process is given a subset of the resources (at any given moment) and NOT a simulation of the whole machine
• Simpler
• Saves a layer of mapping
• Each VM in this case is given a subset of memory, disk space, etc.
• The OS checks for conflicts

Client-Server model

• Microkernel
• Services are moved into user-space processes (e.g. the filesystem)
• The kernel handles message passing mechanisms to make communication possible between user code and services
• Easy to “remote” the message passing (distributed system)
• Resilient: a crash in one module doesn’t compromise the whole system (which can then recover from the crash)
• I/O and HW access must be done into the kernel (spoils a bit the nice client-server model) for example in device drivers
Example: microkernel OS

Microkernel RTOS: In QNX Neutrino, only the most fundamental OS primitives (e.g. signals, timers, scheduling) are handled in the kernel itself. All other components – drivers, file systems, protocol stacks, user applications – run outside the kernel as separate, memory-protected processes.